Grace and Skyler

4/9/21

Professor Jeff Ondich

Basic Authentication

Basic Authentication Story

To begin our examination of HTTP basic authentication, we will use wireshark to see what happens when a client (web browser) interacts with a server that uses basic authentication. Basic authentication is simply a method where the client enters a username and password when making a request to a server. In our study, we, as the client, will enter in a username and password to the website http://cs231.jeffondich.com/basicauth/, which uses basic authentication.

To examine this interaction, we will use Wireshark and study what occurs in this set of client-server interactions, so we want to capture all the interactions on "host 45.79.89.123". Then, we navigated to a web browser and went to http://cs231.jeffondich.com/basicauth/. Upon entering this, immediately, we see packets being sent between the client and server. We also observe that the server running on port 80 and port 58446 is the client. We see that we, as the client, initiate the three-way handshake:

```
1  0.000048426    10.0.2.15        45.79.89.123    TCP     74      53206 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=28061858 TSecr=0 WS=128
2 0.044745154     45.79.89.123     10.0.2.15       TCP     60      80 → 53204 [SYN, ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=1460
3 0.044797906     10.0.2.15        45.79.89.123    TCP     54      53204 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
```

The client sends [SYN], which initializes a connection, then the server responds with [SYN, ACK], which means it acknowledges the client's request and initializes another connection. The third frame is the client acknowledging the server with [ACK]. Then, the client (web browser) makes a GET request to access the /basicauth endpoint, which the server sends a 401 response indicating that is request was not authorized. We believe this is because the client wanted to gain access to this endpoint without completing the basic authentication by entering a valid username and password. So previous to entering a valid username and password, the client cannot access the specified web page.

7 0.044973709 10.0.2.15 45.79.89.123 HTTP 395 GET /basicauth/ HTTP/1.1

8 0.089618550 45.79.89.123 10.0.2.15 HTTP 473 HTTP/1.1 401 Unauthorized (text/html)

Then, we see the closing handshake between the client and server [FIN] finishing and closing the connection, [ACK] acknowledging the closing:

10 5.045217296 10.0.2.15 45.79.89.123 TCP 54 53204 → 80 [FIN, ACK] Seq=1 Ack=1 Win=64240 Len=0

11 5.045434999 45.79.89.123 10.0.2.15 TCP 60 80 → 53204 [ACK] Seq=1 Ack=2 Win=32767 Len=0

12 5.089967912 45.79.89.123 10.0.2.15 TCP 60 80 → 53204 [FIN, ACK] Seq=1 Ack=2 Win=32767 Len=0

13 5.089995226 10.0.2.15 45.79.89.123 TCP 54 53204 → 80 [ACK] Seq=2 Ack=2 Win=64240 Len=0

Following this closing, we see that TCP protocol "keep-alive." At first, we did not understand these packets, but upon looking them up, we think that the server and client are communicating to keep the connection between them going. Thus the client first sends a keep-alive message followed by the server echoing this message:

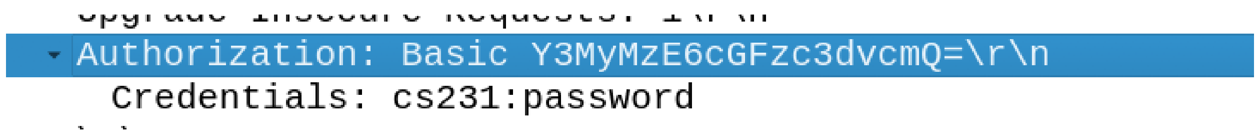
14 10.245463161 10.0.2.15 45.79.89.123 TCP 54 [TCP Keep-Alive] 53206 → 80 [ACK] Seq=341 Ack=420 Win=63821 Len=0

15 10.245714266 45.79.89.123 10.0.2.15 TCP 60 [TCP Keep-Alive ACK] 80 → 53206 [ACK] Seq=420 Ack=342 Win=32427 Len=0

Next, we entered the correct username: cs231 and password: password to the basic authentication protocol and were directed to a web page titled "Index of /basicauth/. After the client enters the password, the browser sends a GET request to the server:

12.072083825 10.0.2.15 45.79.89.123 HTTP 438 GET /basicauth/ HTTP/1.1

In this request, when you look at the contents of the packet/request, you can see the credentials entered by the user looking into Hypertext transfer protocol > Authorization > Credentials: cs231: password. The webpage is sending this user-password concatenation to the server to be validated, and once it is validated, the server acknowledges the request and sends a 200 response which means the request is OK. Here, the client is sending the user-password to the server to check if it is a valid username and password. In this exchange, the web browser sends the user-password encoded in base64. So here, the user-password, cs231:password, is encoded in base 64.



```
Authorization: Basic Y3MyMzE6cGFzc3dvcmQ=\r\n
  Credentials: cs231:password
```

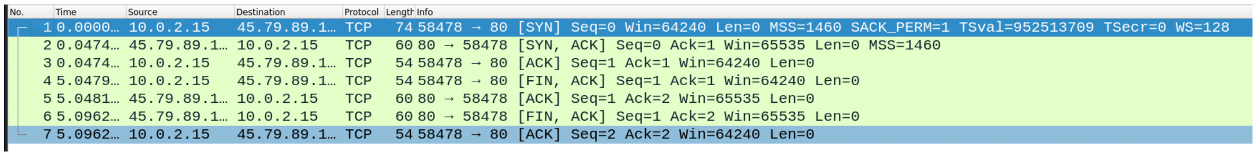
We looked up how basic authentication encoded the user-password and discovered that it is only encoded in base64 and not encrypted or hashed in any way. Thus there is no key phrase. This sequence of events makes sense given the HTTP basic authentication protocol documents, which stated that the user-password would be encoded in base64 and sent to the server for validation. Underneath this authorization tab we see that indeed the user-password is the username concatenated with ':' and password, and this is what is encoded.

Next, in wireshark, we observe the web browser send another GET request to the server for favicon.ico, which we inferred was the web browser requesting to get the icon at the top of the webpage. However, the server could not find this icon and thus sent the 404 response indicating that it could not be found.

19 12.187936347 10.0.2.15 45.79.89.123 HTTP 306 GET /favicon.ico HTTP/1.1

20 12.232809465 45.79.89.123 10.0.2.15 HTTP 401 HTTP/1.1 404 Not Found (text/html)

Next, we decided to visit some of the links in the Index of /basicauth/ endpoint. When we clicked on the top link to go to the homepage, this handshake was shown:



```
No. Time      Source       Destination  Protocol Length Info
  1 0.0000…   10.0.2.15    45.79.89.1…  TCP      74 58478 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=952513709 TSecr=0 WS=128
  2 0.0474…   45.79.89.1…  10.0.2.15    TCP      60 80 → 58478 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
  3 0.0474…   10.0.2.15    45.79.89.1…  TCP      54 58478 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
  4 5.0479…   10.0.2.15    45.79.89.1…  TCP      54 58478 → 80 [FIN, ACK] Seq=1 Ack=1 Win=64240 Len=0
  5 5.0481…   45.79.89.1…  10.0.2.15    TCP      60 80 → 58478 [ACK] Seq=1 Ack=2 Win=65535 Len=0
  6 5.0962…   45.79.89.1…  10.0.2.15    TCP      60 80 → 58478 [FIN, ACK] Seq=1 Ack=2 Win=65535 Len=0
  7 5.0962…   10.0.2.15    45.79.89.1…  TCP      54 58478 → 80 [ACK] Seq=2 Ack=2 Win=64240 Len=0
```

The handshake demonstrates us as the client requesting to open a connection between us and the server, the server opens another connection and acknowledges the client, and so on. When the "secrets" link is clicked on, there is another GET /basicauth/ in wireshark.

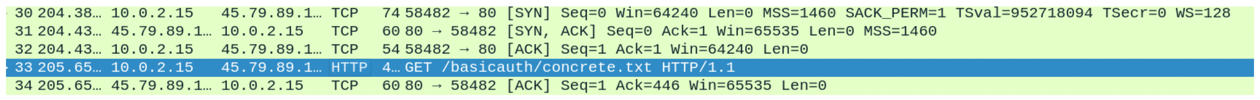
We also clicked on the amateurs link in which we were directed to a page that said, "Amateurs hack systems, professionals hack people" --Bruce Sneider. After clicking on the link in wireshark, we see the client send another GET request to the server:

22 17.905054528 10.0.2.15 45.79.89.123 HTTP 499 GET /basicauth/amateurs.txt HTTP/1.1

The server then sent a response packet to the client indicating that the request went through and that it was OK:

23 17.950357113 45.79.89.123 10.0.2.15 HTTP 375 HTTP/1.1 200 OK (text/plain)

When we clicked on the other two links, pig.txt and concrete.txt, the web browser sent GET requests for those two links, and the server responded with a response 200 OK for text/plain HTTP request.



```
30 204.38… 10.0.2.15    45.79.89.1… TCP   74 58482 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=952718094 TSecr=0 WS=128
31 204.43… 45.79.89.1… 10.0.2.15    TCP   60 80 → 58482 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
32 204.43… 10.0.2.15    45.79.89.1… TCP   54 58482 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
33 205.65… 10.0.2.15    45.79.89.1… HTTP  4… GET /basicauth/concrete.txt HTTP/1.1
34 205.65… 45.79.89.1… 10.0.2.15    TCP   60 80 → 58482 [ACK] Seq=1 Ack=446 Win=65535 Len=0
```

There were many connections between the client and server opened and closed throughout the process of getting to the web page, entering the username and password, and clicking on the inner URLs. The client is always asking the server if it can do something; if the server acknowledges it and opens a new connection, the client acknowledges, and then there is an HTTP GET request, and the web page is changed. Then the server acknowledges with [ACK].

After this, we came away with one question. First is why do the server and client close the connection with the [FIN] but then use the keep-alive protocol to keep the connection alive?